多态的意义：

1） 没有使用多态以前，每种不同的类型需要分别处理：

public static void main(String[] args) {

Student s1 = new Student();

Student s2 = new Student();

Student s3 = new Student();

Student s4 = new Student();

Teacher t1 = new Teacher();

Teacher t2 = new Teacher();

Fighter f1 = new Fighter();

Fighter f2 = new Fighter();

aa(new Student[]{s1,s2,s3,s4}, new Teacher[]{t1,t2},

new Fighter[]{f1,f2});

}

public static void aa(Student[] students, Teacher[] teachers,

Fighter[] fighters) {

for(int i = 0; i < students.length; i++) {

students[i].fight();

}

for(int i = 0; i < teachers.length; i++) {

teachers[i].fight();

}

for(int i = 0; i < fighters.length; i++) {

fighters[i].fight();

}

}

1. 使用了多态以后，可以用父类的类型对子类的对象进行统一处理：

public static void main(String[] args) {

Person s1 = new Student();

Person s2 = new Student();

Person s3 = new Student();

Person s4 = new Student();

Person t1 = new Teacher();

Person t2 = new Teacher();

Person f1 = new Fighter();

Person f2 = new Fighter();

Person c1 = new Coder();

aa(new Person[]{s1,s2,s3,s4,t1,t2,f1,f2,c1});

}

public static void aa(Person[] person) {

for(int i = 0; i < person.length; i++) {

person[i].fight();// 实际类型的方法

}

}

## 抽象类 abstract class

当一个方法不需要具体代码时，可以把这个方法设计为抽象方法

抽象方法的格式： abstract 方法返回值类型 方法名(方法参数...) ;

抽象类的格式 : abstract class 类名 { ... }

// 一个类如果包含了抽象方法，这个类必须是抽象类

public abstract class Person {

// 定义抽象方法，不需要有方法体

abstract void fight();

}

注意：

如果子类继承了有抽象方法的父类，有两种选择：

1. 由子类自己来实现抽象方法
2. 把子类变成抽象类，由下一代来实现抽象方法

特征：

1. 抽象类不能够实例化 不能 new 抽象类();
2. 抽象类可以有构造方法
3. 抽象类中可以有带实现的方法

// 一个类如果包含了抽象方法，这个类必须是抽象类

public abstract class Person {

// 定义抽象方法，不需要有方法体

abstract void fight();

// 可以有带实现的方法

void method1() {

System.out.println("实现");

}

// 可以有构造

public Person() {

System.out.println("Person的构造");

}

}

不能够new抽象类：

// 抽象类不能被实例化

// Person p1 = new Person(); 错误

// Person f1 = new Fighter(); 错误

## 接口 （interface)

当抽象类中所有的方法都是抽象方法时，可以把抽象类声明为一个接口

语法:

public interface 接口名 {

抽象方法1；

抽象方法2；

...

}

接口中不能有别的方法和构造方法

步骤1：抽取共同行为到接口(定义接口)

public interface Fight {

public abstract void fight();

}

// 其中abstract 可以省略, public可以省略：接口中所有方法都是公共的和抽象的

步骤2： 实现接口

语法：

实现

public class 类名 implements 接口名字 {

// 实现接口中所有抽象方法

}

public class Student implements Fight {

// 实现了接口中的抽象方法

public void fight() {

System.out.println("砖头、木棍、拳头");

}

}

public class Dog implements Fight{

public void fight() {

System.out.println("咬、叫唤");

}

}

步骤3：通过接口实现统一调用

接口类型 f = new 实现了接口的类（）；

public static void main(String[] args) {

// 实现了接口的对象，可以赋值给接口类型的变量

Fight s1 = new Student();

Fight s2 = new Student();

Fight d1 = new Dog();

Fight d2 = new Dog();

// 统一处理

Fight[] all = new Fight[]{s1,s2,d1,d2};

for(int i = 0; i<all.length; i++) {

all[i].fight();

}

}

### 接口的特点：

一个类可以实现多个接口

皮鞋, 既能当皮鞋用，还能当剃须刀用，还能当电吹风用

// 剃须刀接口

public interface Tixudao {

// 刮胡子的方法

public void gua();

}

// 电吹风

public interface Chuifeng {

// 吹的方法

public void chui();

}

public interface Shoes {

// 皮鞋接口提供走路方法

public void walk();

}

// 周星驰的皮鞋

public class ZhouShoes implements Tixudao, Chuifeng, Shoes {

public void gua() {

System.out.println("刮胡子");

}

public void chui() {

System.out.println("吹风");

}

public void walk() {

System.out.println("走路");

}

public static void main(String[] args) {

// 当剃须刀用

// Tixudao txd = new ZhouShoes();

// txd.gua();

// 当电吹风用

// Chuifeng cf = new ZhouShoes();

// cf.chui();

// 当鞋来用

Shoes s = new ZhouShoes();

s.walk();

}

}

一个对象可以同时继承父类和实现多个接口

public ZhouShoes extends Shoes implements Tixudao, Chuifeng {

}

1. 一个子类又要继承父类 又要实现接口 该怎么用？

不管是子类继承父类还是 子类实现接口，都表示的是“是”的关系

可以把子类对象，赋值给 它“是”的类型

Shoes s = new ZhouShoes(); 是这个类型，可以赋值

Tixudao t = new ZhouShoes(); 是这个类型，可以赋值

Chuifeng c = new ZhouShoes(); 是这个类型，可以赋值

Person p = new ZhouShoes(); 不是Person类型，不能赋值

2.如果父类是一个接口的实现 那么它的子类该怎么用接口里的方法呢？

// 吃饭接口

public interface Eat {

public abstract void eat();

}

public abstract class Person implements Eat{

// 类里有一个抽象的eat方法

// public void eat() {

// System.out.println("吃饭");

// }

}

public class Student extends Person {

public void eat() {

System.out.println("吃饭");

}

public static void main(String[] args) {

// Student s = new Student();

// s.eat();

Person s = new Student();

s.eat();

// 父类里定义了抽象的eat方法（通过接口有了这个定义）

// 调用时，是用了子类对抽象eat方法的实现

}

}

## 类型转换

class Person {}

class Student extends Person {

double fee; // 学费

}

class Teacher extends Person {

double salary; // 工资

double bouns;// 奖金

}

public class Test {

public static void main(String[] args) {

Person p = new Student();

// (类型) ==》 强制转换

Student s = (Student) p; // 如何将它转换为实际类型

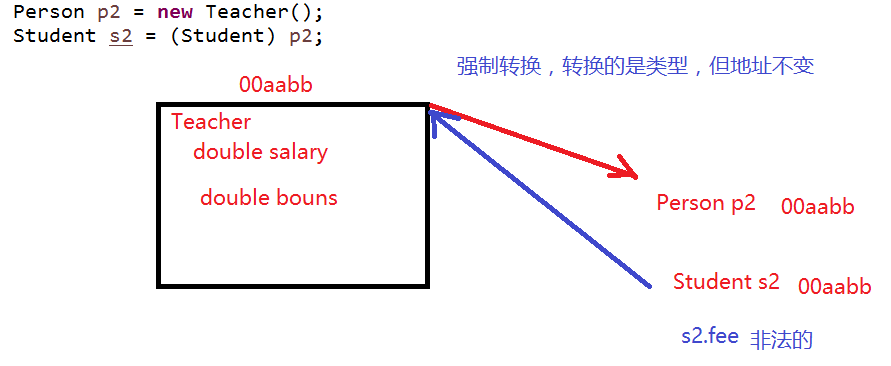
Person p2 = new Teacher();

Student s2 = (Student) p2; // 编译不会出错，运行会出错

System.out.println(s2.fee);

}

}



结论：

可以使用强制转换运算符来对不同的类型进行转换。

但是只能转换为 它的实际的类型，或是父类类型，或是接口类型

不能把他转换为没有关系的其他类型

// 演示强制转换

public static void main(String[] args) {

// 只能被强制转换为接口类型，自身类型，父类类型

// 能调用哪些方法，是由变量的类型决定的

ZhouShoes a = new ZhouShoes();

a.chui();

a.gua();

a.walk();

Tixudao b = (Tixudao)a;

Chuifeng c = (Chuifeng)a;

Shoes s = (Shoes) a;

}